BOOK 7: THE DOM
Chapter 1: Selecting Elements

WHAT IS THE DOM

The DOM (Document Object Model) is how JavaScript sees a web page.

When a browser loads HTML, it creates a tree structure in memory. Each HTML element becomes a node in this tree. JavaScript can read and modify this tree.

HTML file:

```
<!DOCTYPE html>
<html>
<head>
 <title>My Page</title>
</head>
<body>
 <h1>Hello</h1>
 <p>Welcome to my page</p>
</body>
</html>
```

The browser creates a tree:

```
document
 |
 html
 |
 +-- head
 |  |
 |  title ("My Page")
 |
 +-- body
 |
 +-- h1 ("Hello")
```

|
+-- p ("Welcome to my page")

JavaScript can access any part of this tree.

THE DOCUMENT OBJECT

document is the entry point to the DOM:

```
console.log(document)
console.log(document.title)
console.log(document.body)
console.log(document.head)
```

document.title changes the page title:

```
document.title = "New Title"
```

Look at your browser tab. The title changed.

GETELEMENTBYID

Select an element by its id attribute:

HTML:

```
<h1 id="main-title">Welcome</h1>
<p id="intro">This is my page</p>
```

JavaScript:

```
let title = document.getElementById("main-title")
let intro = document.getElementById("intro")
```

```
console.log(title)
console.log(intro)

console.log(title.textContent)
```

Output: Welcome

getElementById returns a single element or null if not found:

```
let missing = document.getElementById("does-not-exist")
console.log(missing)
```

Output: null

IDs must be unique on a page. Only one element should have each id.

GETELEMENTSBYCLASSNAME

Select elements by their class attribute:

HTML:

```
<p class="highlight">First paragraph</p>
<p class="highlight">Second paragraph</p>
<p>Third paragraph</p>
```

JavaScript:

```
let highlights = document.getElementsByClassName("highlight")

console.log(highlights)
console.log(highlights.length)
```

Output:
HTMLCollection [p.highlight, p.highlight]
2

getElementsByClassName returns an HTMLCollection (like an array).

Loop through the results:

```
for (let element of highlights) {
 console.log(element.textContent)
}
```

Output:
First paragraph
Second paragraph

GETELEMENTSBYTAGNAME

Select elements by their tag name:

HTML:

```
<ul>
 <li>Apple</li>
 <li>Banana</li>
 <li>Cherry</li>
</ul>
```

JavaScript:

```
let items = document.getElementsByTagName("li")

console.log(items.length)

for (let item of items) {
 console.log(item.textContent)
}
```

Output:

3
Apple
Banana
Cherry

Common tag names: div, p, span, a, img, input, button, ul, li, h1, h2, h3

QUERYSELECTOR

Select the first element matching a CSS selector:

HTML:

```
<div class="container">
 <h1>Title</h1>
 <p class="intro">First paragraph</p>
 <p>Second paragraph</p>
</div>
```

JavaScript:

```
let container = document.querySelector(".container")
let title = document.querySelector("h1")
let intro = document.querySelector(".intro")
let firstP = document.querySelector("p")

console.log(container)
console.log(title.textContent)
console.log(intro.textContent)
console.log(firstP.textContent)
```

querySelector uses CSS selector syntax:

```
document.querySelector("#main-title")
document.querySelector(".highlight")
document.querySelector("p")
```

```
document.querySelector("div.container")
document.querySelector("ul li")
document.querySelector("input[type='text']")
```

querySelector returns the first match or null.

QUERYSELECTORALL

Select all elements matching a CSS selector:

HTML:

```
<ul>
 <li class="item">Apple</li>
 <li class="item">Banana</li>
 <li class="item">Cherry</li>
</ul>
```

JavaScript:

```
let items = document.querySelectorAll(".item")

console.log(items.length)

items.forEach(item => {
 console.log(item.textContent)
})
```

Output:
3
Apple
Banana
Cherry

querySelectorAll returns a NodeList. NodeList has forEach.
HTMLCollection does not.

Complex selectors:

```
let links = document.querySelectorAll("nav a")
let buttons = document.querySelectorAll("button.primary")
let inputs = document.querySelectorAll("input[required]")
let oddItems = document.querySelectorAll("li:nth-child(odd)")
```

HTMLCOLLECTION VS NODELIST

getElementsByClassName and getElementsByTagName return HTMLCollection.
querySelectorAll returns NodeList.

Differences:

HTMLCollection:
- Live (updates when DOM changes)
- No forEach method
- Access by index or name

NodeList:
- Static (snapshot at selection time)
- Has forEach method
- Access by index only

HTMLCollection to array:

```
let highlights = document.getElementsByClassName("highlight")
let array = Array.from(highlights)

array.forEach(element => {
 console.log(element.textContent)
})
```

SELECTING WITHIN ELEMENTS

Select elements within a specific parent:

HTML:

```
<div id="sidebar">
  <p>Sidebar paragraph</p>
</div>
<div id="main">
  <p>Main paragraph</p>
</div>
```

JavaScript:

```
let sidebar = document.getElementById("sidebar")
let main = document.getElementById("main")

let sidebarP = sidebar.querySelector("p")
let mainP = main.querySelector("p")

console.log(sidebarP.textContent)
console.log(mainP.textContent)
```

Output:
Sidebar paragraph
Main paragraph

All selection methods work on any element, not just document.

CHECKING IF ELEMENT EXISTS

Always check before using:

```
let element = document.getElementById("maybe-exists")

if (element) {
```

```
 console.log(element.textContent)
} else {
 console.log("Element not found")
}
```

Or use optional chaining:

```
let text = document.getElementById("maybe-exists")?.textContent
console.log(text)
```

PRACTICAL EXAMPLES

Find all links on a page:

```
let links = document.querySelectorAll("a")

links.forEach(link => {
 console.log(link.href)
})
```

Count elements:

```
let paragraphs = document.querySelectorAll("p")
let divs = document.querySelectorAll("div")
let images = document.querySelectorAll("img")

console.log(`Paragraphs: ${paragraphs.length}`)
console.log(`Divs: ${divs.length}`)
console.log(`Images: ${images.length}`)
```

Find element by attribute:

```
let submitButton =
document.querySelector("button[type='submit']")
let emailInput = document.querySelector("input[name='email']")
let externalLinks =
document.querySelectorAll("a[target='_blank']")
```

PRACTICE

Select an element by id.

Select elements by class name.

Use querySelector with different selectors.

Loop through querySelectorAll results.

Select elements within a specific parent element.

WHAT YOU LEARNED

You learned the DOM is a tree structure of HTML.
You learned getElementById selects by id.
You learned getElementsByClassName selects by class.
You learned querySelector selects the first match.
You learned querySelectorAll selects all matches.
You learned to select within specific elements.

The next chapter teaches you how to modify elements.
BOOK 7: THE DOM
Chapter 2: Modifying Elements

TEXTCONTENT

Change the text inside an element:

HTML:

```
<h1 id="title">Original Title</h1>
```

JavaScript:

```
let title = document.getElementById("title")

console.log(title.textContent)

title.textContent = "New Title"

console.log(title.textContent)
```

Output:
Original Title
New Title

textContent replaces all text inside the element.

INNERTEXT VS TEXTCONTENT

Both get text, but they differ:

HTML:

```
<p id="text">Hello <span style="display: none">hidden</
span> world</p>
```

JavaScript:

```
let element = document.getElementById("text")

console.log(element.textContent)
console.log(element.innerText)
```

Output:

Hello hidden world
Hello world

textContent returns all text including hidden.
innerText returns only visible text.

Use textContent. It is faster and more predictable.

INNERHTML

Change the HTML inside an element:

HTML:

```
<div id="container">
 <p>Old content</p>
</div>
```

JavaScript:

```
let container = document.getElementById("container")

console.log(container.innerHTML)

container.innerHTML = "<h2>New heading</h2><p>New paragraph</p>"

console.log(container.innerHTML)
```

innerHTML parses HTML. textContent treats everything as plain text.

Security warning: Never use innerHTML with user input. It can execute malicious scripts.

```
let userInput = "<script>alert('hacked')</script>"
```

container.innerHTML = userInput

container.textContent = userInput

The second line is safe. The first is dangerous.

CHANGING ATTRIBUTES

Get and set attributes:

HTML:

```
<a id="link" href="https://example.com">Click me</a>
<img id="image" src="photo.jpg" alt="A photo">
```

JavaScript:

```
let link = document.getElementById("link")
let image = document.getElementById("image")

console.log(link.getAttribute("href"))

link.setAttribute("href", "https://new-site.com")
link.setAttribute("target", "_blank")

image.setAttribute("src", "new-photo.jpg")
image.setAttribute("alt", "A new photo")
```

Common attributes: href, src, alt, title, class, id, name, value, type

DIRECT ATTRIBUTE ACCESS

Many attributes have direct properties:

```
let link = document.getElementById("link")

console.log(link.href)
console.log(link.id)

link.href = "https://another-site.com"

let image = document.getElementById("image")

console.log(image.src)
console.log(image.alt)

image.src = "different.jpg"

let input = document.getElementById("email")

console.log(input.value)
console.log(input.type)
console.log(input.name)
```

REMOVING ATTRIBUTES

```
let link = document.getElementById("link")

link.removeAttribute("target")
```

Check if attribute exists:

```
if (link.hasAttribute("target")) {
 console.log("Has target attribute")
} else {
 console.log("No target attribute")
}
```

CHANGING CLASSES

HTML:

```
<div id="box" class="small blue">Box</div>
```

JavaScript:

```
let box = document.getElementById("box")

console.log(box.className)

box.className = "large red"

console.log(box.className)
```

Output:
small blue
large red

className replaces all classes. Use classList for more control.

CLASSLIST

Add, remove, and toggle classes:

```
let box = document.getElementById("box")

box.classList.add("highlight")

box.classList.remove("small")

box.classList.toggle("active")

console.log(box.classList.contains("highlight"))
```

toggle adds the class if missing, removes if present.

Add multiple classes:

box.classList.add("one", "two", "three")

box.classList.remove("one", "two")

Replace a class:

box.classList.replace("old-class", "new-class")

CHANGING STYLES

Inline styles with the style property:

let box = document.getElementById("box")

box.style.backgroundColor = "red"
box.style.color = "white"
box.style.padding = "20px"
box.style.border = "2px solid black"
box.style.borderRadius = "10px"

CSS properties with hyphens become camelCase:

background-color becomes backgroundColor
font-size becomes fontSize
border-radius becomes borderRadius
margin-top becomes marginTop

Get computed style:

let computedStyle = getComputedStyle(box)

console.log(computedStyle.backgroundColor)
console.log(computedStyle.width)

getComputedStyle returns the final calculated values.

MULTIPLE STYLE CHANGES

Use cssText for multiple styles:

```
let box = document.getElementById("box")

box.style.cssText = "background: blue; color: white; padding:
10px;"
```

Or add a class instead:

CSS:

```
.highlight {
 background: yellow;
 color: black;
 padding: 10px;
}
```

JavaScript:

```
box.classList.add("highlight")
```

Adding classes is better than inline styles for maintainability.

HIDING AND SHOWING ELEMENTS

Hide an element:

```
let element = document.getElementById("box")

element.style.display = "none"
```

Show an element:

element.style.display = "block"

Or use visibility:

element.style.visibility = "hidden"

element.style.visibility = "visible"

Difference:
display: none removes the element from layout.
visibility: hidden hides but keeps the space.

PRACTICAL EXAMPLES

Toggle dark mode:

```
let body = document.body

function toggleDarkMode() {
 body.classList.toggle("dark-mode")
}
```

Update counter:

```
let counter = document.getElementById("counter")
let count = 0

function increment() {
 count++
 counter.textContent = count
}
```

Change link destination:

```
let links = document.querySelectorAll("a")

links.forEach(link => {
 if (link.href.includes("http://")) {
 link.href = link.href.replace("http://", "https://")
 }
})
```

Add loading state:

```
let button = document.getElementById("submit-btn")

function startLoading() {
 button.textContent = "Loading..."
 button.classList.add("loading")
 button.setAttribute("disabled", "true")
}

function stopLoading() {
 button.textContent = "Submit"
 button.classList.remove("loading")
 button.removeAttribute("disabled")
}
```

PRACTICE

Change the text of an element.

Change an element's background color.

Add and remove classes.

Toggle visibility of an element.

Modify an attribute value.

WHAT YOU LEARNED

You learned textContent changes text inside elements.
You learned innerHTML changes HTML content.
You learned setAttribute and getAttribute modify attributes.
You learned classList for adding and removing classes.
You learned the style property for inline styles.
You learned to hide and show elements.

The next chapter teaches you how to create and remove elements.
BOOK 7: THE DOM
Chapter 3: Creating and Removing Elements

CREATING ELEMENTS

Use document.createElement:

```
let paragraph = document.createElement("p")

paragraph.textContent = "This is a new paragraph"

console.log(paragraph)
```

The element exists but is not on the page yet. You must add it to the DOM.

ADDING ELEMENTS TO THE PAGE

appendChild adds to the end:

```
let container = document.getElementById("container")
```

```
let paragraph = document.createElement("p")
paragraph.textContent = "New paragraph"

container.appendChild(paragraph)
```

The paragraph now appears inside the container.

Create and configure before adding:

```
let link = document.createElement("a")
link.href = "https://example.com"
link.textContent = "Visit Example"
link.target = "_blank"

container.appendChild(link)
```

APPEND VS APPENDCHILD

append is newer and more flexible:

```
let container = document.getElementById("container")

container.append("Plain text")

let p = document.createElement("p")
p.textContent = "Paragraph"
container.append(p)

container.append("Text", p, "More text")
```

append accepts strings and multiple elements.
appendChild only accepts one node and returns it.

PREPEND

Add to the beginning:

```
let container = document.getElementById("container")

let first = document.createElement("p")
first.textContent = "I am first"

container.prepend(first)
```

INSERTBEFORE

Insert before a specific element:

HTML:

```
<ul id="list">
 <li>First</li>
 <li id="second">Second</li>
 <li>Third</li>
</ul>
```

JavaScript:

```
let list = document.getElementById("list")
let second = document.getElementById("second")

let newItem = document.createElement("li")
newItem.textContent = "New Item"

list.insertBefore(newItem, second)
```

Result:

```
<ul id="list">
 <li>First</li>
```

```
 <li>New Item</li>
 <li id="second">Second</li>
 <li>Third</li>
</ul>
```

INSERTADJACENTHTML

Insert HTML at specific positions:

```
let container = document.getElementById("container")

container.insertAdjacentHTML("beforebegin", "<p>Before container</p>")

container.insertAdjacentHTML("afterbegin", "<p>First inside container</p>")

container.insertAdjacentHTML("beforeend", "<p>Last inside container</p>")

container.insertAdjacentHTML("afterend", "<p>After container</p>")
```

Positions:
beforebegin - before the element
afterbegin - first child inside
beforeend - last child inside
afterend - after the element

INSERTADJACENTELEMENT

Same positions but with an element:

```
let newParagraph = document.createElement("p")
```

```
newParagraph.textContent = "Inserted paragraph"

let container = document.getElementById("container")
container.insertAdjacentElement("afterbegin", newParagraph)
```

CLONING ELEMENTS

Clone an existing element:

```
let original = document.getElementById("template")

let clone = original.cloneNode(false)

let deepClone = original.cloneNode(true)
```

cloneNode(false) clones only the element.
cloneNode(true) clones the element and all its children.

HTML:

```
<div id="template">
 <h2>Title</h2>
 <p>Content</p>
</div>
```

JavaScript:

```
let template = document.getElementById("template")

let shallow = template.cloneNode(false)
console.log(shallow.innerHTML)

let deep = template.cloneNode(true)
console.log(deep.innerHTML)
```

Output:
(empty)

<h2>Title</h2> Content

REMOVING ELEMENTS

Remove an element:

```
let element = document.getElementById("to-remove")

element.remove()
```

The element is gone from the page.

Older method using parent:

```
let element = document.getElementById("to-remove")
let parent = element.parentNode

parent.removeChild(element)
```

remove() is simpler. Use it.

REPLACING ELEMENTS

Replace one element with another:

```
let oldElement = document.getElementById("old")

let newElement = document.createElement("div")
newElement.textContent = "I am new"

oldElement.replaceWith(newElement)
```

Or using parent:

```
let parent = oldElement.parentNode
parent.replaceChild(newElement, oldElement)
```

CLEARING ALL CHILDREN

Remove all children from an element:

```
let container = document.getElementById("container")

container.innerHTML = ""

while (container.firstChild) {
 container.removeChild(container.firstChild)
}

container.replaceChildren()
```

All three methods clear the container.

BUILDING LISTS DYNAMICALLY

```
let fruits = ["Apple", "Banana", "Cherry", "Date"]

let list = document.getElementById("fruit-list")

fruits.forEach(fruit => {
 let item = document.createElement("li")
 item.textContent = fruit
 list.appendChild(item)
})
```

Result:

```
<ul id="fruit-list">
```

```
 <li>Apple</li>
 <li>Banana</li>
 <li>Cherry</li>
 <li>Date</li>
</ul>
```

DOCUMENT FRAGMENT

For better performance when adding many elements:

```
let fruits = ["Apple", "Banana", "Cherry", "Date", "Elderberry"]

let fragment = document.createDocumentFragment()

fruits.forEach(fruit => {
 let item = document.createElement("li")
 item.textContent = fruit
 fragment.appendChild(item)
})

document.getElementById("list").appendChild(fragment)
```

DocumentFragment is a lightweight container. Adding elements to it does not cause page reflows. Add everything to the fragment first, then add the fragment to the page once.

PRACTICAL EXAMPLES

Add item to todo list:

```
function addTodo(text) {
 let list = document.getElementById("todo-list")

 let item = document.createElement("li")
```

```
 item.textContent = text

 let deleteBtn = document.createElement("button")
 deleteBtn.textContent = "Delete"
 deleteBtn.onclick = function() {
 item.remove()
 }

 item.appendChild(deleteBtn)
 list.appendChild(item)
}

addTodo("Learn JavaScript")
addTodo("Build a project")
```

Build a table from data:

```
let users = [
 { name: "Alice", age: 25 },
 { name: "Bob", age: 30 },
 { name: "Charlie", age: 35 }
]

let table = document.getElementById("user-table")
let tbody = table.querySelector("tbody")

users.forEach(user => {
 let row = document.createElement("tr")

 let nameCell = document.createElement("td")
 nameCell.textContent = user.name

 let ageCell = document.createElement("td")
 ageCell.textContent = user.age

 row.appendChild(nameCell)
 row.appendChild(ageCell)
 tbody.appendChild(row)
})
```

Create notification:

```
function showNotification(message) {
 let notification = document.createElement("div")
 notification.className = "notification"
 notification.textContent = message

 document.body.appendChild(notification)

 setTimeout(() => {
 notification.remove()
 }, 3000)
}

showNotification("Settings saved!")
```

PRACTICE

Create a new element and add it to the page.

Clone an element and add the clone.

Remove an element from the page.

Build a list from an array.

Create a function that adds and removes elements.

WHAT YOU LEARNED

You learned createElement creates new elements.
You learned appendChild and append add elements.
You learned insertBefore for specific positioning.
You learned insertAdjacentHTML for HTML strings.

You learned cloneNode to copy elements.
You learned remove to delete elements.
You learned DocumentFragment for performance.

The next chapter teaches you about attributes and data attributes.

BOOK 7: THE DOM

Chapter 4: Attributes and Data Attributes

HTML ATTRIBUTES

Every HTML element can have attributes:

```
<a href="https://example.com" target="_blank" title="Visit Example">Link</a>

<img src="photo.jpg" alt="A photo" width="300" height="200">

<input type="text" name="email" placeholder="Enter email" required>
```

Attributes provide extra information about elements.

GETTING ATTRIBUTES

Use getAttribute:

HTML:

```
<a id="link" href="https://example.com" target="_blank">Click me</a>
```

JavaScript:

```
let link = document.getElementById("link")

let href = link.getAttribute("href")
let target = link.getAttribute("target")

console.log(href)
console.log(target)
```

Output:
https://example.com
_blank

getAttribute returns null if the attribute does not exist:

```
let missing = link.getAttribute("data-custom")
console.log(missing)
```

Output: null

SETTING ATTRIBUTES

Use setAttribute:

```
let link = document.getElementById("link")

link.setAttribute("href", "https://new-site.com")
link.setAttribute("title", "New title")
link.setAttribute("rel", "noopener")
```

setAttribute creates the attribute if it does not exist.

REMOVING ATTRIBUTES

Use removeAttribute:

```
let link = document.getElementById("link")

link.removeAttribute("target")
link.removeAttribute("title")
```

The attributes are gone from the element.

CHECKING ATTRIBUTES

Use hasAttribute:

```
let link = document.getElementById("link")

if (link.hasAttribute("target")) {
 console.log("Opens in new tab")
} else {
 console.log("Opens in same tab")
}
```

DIRECT PROPERTY ACCESS

Many attributes have corresponding properties:

```
let link = document.getElementById("link")

console.log(link.href)
console.log(link.target)
console.log(link.id)

link.href = "https://another-site.com"
link.target = "_self"
```

```
let image = document.getElementById("photo")

console.log(image.src)
console.log(image.alt)
console.log(image.width)

let input = document.getElementById("email")

console.log(input.value)
console.log(input.type)
console.log(input.placeholder)
```

ATTRIBUTE VS PROPERTY

They are related but different:

HTML:

```
<input id="name" type="text" value="Alice">
```

JavaScript:

```
let input = document.getElementById("name")

console.log(input.getAttribute("value"))
console.log(input.value)

input.value = "Bob"

console.log(input.getAttribute("value"))
console.log(input.value)
```

Output:
Alice
Alice
Alice
Bob

The attribute is the initial value in HTML.
The property is the current value in JavaScript.

For most cases, use properties. They reflect current state.

BOOLEAN ATTRIBUTES

Some attributes are boolean (present or absent):

```
<input type="checkbox" checked>
<button disabled>Cannot click</button>
<input type="text" required>
```

In JavaScript, use the property:

```
let checkbox = document.getElementById("agree")

console.log(checkbox.checked)
checkbox.checked = true
checkbox.checked = false

let button = document.getElementById("submit")

console.log(button.disabled)
button.disabled = true
button.disabled = false
```

Do not use setAttribute for boolean attributes:

```
checkbox.setAttribute("checked", "false")

checkbox.checked = false
```

DATA ATTRIBUTES

Custom attributes for storing data:

HTML:

```
<div id="user" data-id="123" data-name="Alice" data-
role="admin">
 User info
</div>
```

Data attributes start with data- followed by any name.

They are valid HTML. Use them to attach custom data to elements.

ACCESSING DATA ATTRIBUTES

Use dataset:

```
let user = document.getElementById("user")

console.log(user.dataset.id)
console.log(user.dataset.name)
console.log(user.dataset.role)
```

Output:
123
Alice
admin

dataset is an object containing all data attributes.

Modify data attributes:

```
user.dataset.id = "456"
user.dataset.name = "Bob"
user.dataset.active = "true"
```

Delete a data attribute:

delete user.dataset.role

DATA ATTRIBUTE NAMING

HTML uses kebab-case. JavaScript uses camelCase.

HTML:

JavaScript:

element.dataset.userId
element.dataset.createdAt
element.dataset.isActive

data-user-id becomes dataset.userId
data-created-at becomes dataset.createdAt
data-is-active becomes dataset.isActive

USING DATA ATTRIBUTES

Store IDs for server communication:

<button data-product-id="42" class="add-to-cart">Add to Cart</button>

let buttons = document.querySelectorAll(".add-to-cart")

buttons.forEach(button => {

```
 button.onclick = function() {
 let productId = this.dataset.productId
 addToCart(productId)
 }
})
```

Store configuration:

```
<div id="slider" data-speed="500" data-autoplay="true" data-
interval="3000">

let slider = document.getElementById("slider")
let config = {
 speed: parseInt(slider.dataset.speed),
 autoplay: slider.dataset.autoplay === "true",
 interval: parseInt(slider.dataset.interval)
}
```

Store state:

```
<li data-completed="false">Buy groceries</li>

let item = document.querySelector("li")

function toggleComplete() {
 let isCompleted = item.dataset.completed === "true"
 item.dataset.completed = !isCompleted
}
```

SELECTING BY DATA ATTRIBUTE

Use attribute selectors:

```
let activeUsers = document.querySelectorAll("[data-active='true']")

let adminUser = document.querySelector("[data-role='admin']")
```

```
let allWithId = document.querySelectorAll("[data-id]")

let highPriority = document.querySelectorAll("[data-
priority='high']")
```

PRACTICAL EXAMPLES

Tab system:

HTML:

```
<div class="tabs">
 <button data-tab="home" class="active">Home</button>
 <button data-tab="about">About</button>
 <button data-tab="contact">Contact</button>
</div>

<div data-content="home" class="tab-content active">Home
content</div>
<div data-content="about" class="tab-content">About content</
div>
<div data-content="contact" class="tab-content">Contact
content</div>
```

JavaScript:

```
let tabButtons = document.querySelectorAll("[data-tab]")

tabButtons.forEach(button => {
 button.onclick = function() {
 let tabName = this.dataset.tab

 document.querySelectorAll("[data-tab]").forEach(btn => {
 btn.classList.remove("active")
 })
 this.classList.add("active")
```

```
 document.querySelectorAll("[data-content]").forEach(content => {
 content.classList.remove("active")
 })
 document.querySelector(`[data-content='${tabName}']`).classList.add("active")
 }
})
```

Product listing:

HTML:

```
<div class="product" data-id="1" data-price="29.99" data-category="electronics">
 <h3>Headphones</h3>
 <button class="buy">Buy Now</button>
</div>
```

JavaScript:

```
let products = document.querySelectorAll(".product")

products.forEach(product => {
 let buyBtn = product.querySelector(".buy")

 buyBtn.onclick = function() {
 let id = product.dataset.id
 let price = parseFloat(product.dataset.price)
 let category = product.dataset.category

 console.log(`Buying product ${id} for $${price} from ${category}`)
 }
})
```

Filter by category:

```
function filterByCategory(category) {
 let products = document.querySelectorAll(".product")
```

```
 products.forEach(product = > {
 if (category = = = "all" || product.dataset.category = = =
category) {
 product.style.display = "block"
 } else {
 product.style.display = "none"
 }
 })
}
```

PRACTICE

Get and set attributes on an element.

Use data attributes to store custom information.

Select elements by data attribute value.

Build a feature using data attributes for state.

Convert between data attribute names and dataset properties.

WHAT YOU LEARNED

You learned getAttribute, setAttribute, and removeAttribute.
You learned the difference between attributes and properties.
You learned boolean attributes use properties not setAttribute.
You learned data attributes store custom data.
You learned dataset provides camelCase access.
You learned to select by data attributes.

The next chapter teaches you about styles and classes in depth.
BOOK 7: THE DOM
Chapter 5: Styles and Classes

INLINE STYLES

The style property sets inline styles:

```
let box = document.getElementById("box")

box.style.backgroundColor = "blue"
box.style.color = "white"
box.style.padding = "20px"
box.style.margin = "10px"
```

This adds style attributes directly to the element:

```
<div id="box" style="background-color: blue; color: white;
padding: 20px; margin: 10px;">
```

CSS PROPERTY NAMES IN JAVASCRIPT

CSS uses kebab-case. JavaScript uses camelCase.

background-color becomes backgroundColor
font-size becomes fontSize
border-radius becomes borderRadius
margin-top becomes marginTop
z-index becomes zIndex
text-decoration becomes textDecoration

Common conversions:

```
element.style.fontSize = "16px"
element.style.fontWeight = "bold"
```

```
element.style.textAlign = "center"
element.style.borderBottom = "1px solid black"
element.style.boxShadow = "0 2px 4px rgba(0,0,0,0.1)"
```

READING INLINE STYLES

Get the current inline style value:

```
let box = document.getElementById("box")

box.style.backgroundColor = "red"

console.log(box.style.backgroundColor)
```

Output: red

But this only reads inline styles:

CSS:

```
#box {
 background-color: blue;
}
```

JavaScript:

```
let box = document.getElementById("box")

console.log(box.style.backgroundColor)
```

Output: (empty string)

The style property does not read styles from CSS files.

GETCOMPUTEDSTYLE

Get the actual computed style:

let box = document.getElementById("box")

let computed = getComputedStyle(box)

console.log(computed.backgroundColor)
console.log(computed.width)
console.log(computed.fontSize)
console.log(computed.display)

getComputedStyle returns all final computed values including CSS file styles, browser defaults, and inheritance.

Values are read-only:

computed.backgroundColor = "red"

The computed style does not change.

CSSTEXT

Set multiple styles at once:

let box = document.getElementById("box")

box.style.cssText = "background: blue; color: white; padding: 20px; border-radius: 10px;"

Warning: cssText replaces all existing inline styles.

let box = document.getElementById("box")
box.style.backgroundColor = "red"

box.style.cssText = "color: white;"

console.log(box.style.backgroundColor)

Output: (empty)

The backgroundColor is gone because cssText replaced everything.

REMOVING STYLES

Set to empty string:

box.style.backgroundColor = ""

Or use removeProperty:

box.style.removeProperty("background-color")

Note: removeProperty uses the CSS name (kebab-case), not the JavaScript name.

CLASSNAME

The className property:

let box = document.getElementById("box")

console.log(box.className)

box.className = "large blue rounded"

console.log(box.className)

className is a string of all classes separated by spaces.

Replace all classes:

box.className = "new-class"

Add a class (awkward):

box.className = box.className + " extra-class"

className is inconvenient for managing multiple classes. Use classList instead.

CLASSLIST

Better class management:

let box = document.getElementById("box")

box.classList.add("highlight")

box.classList.remove("old-class")

box.classList.toggle("active")

let hasClass = box.classList.contains("highlight")

ADD MULTIPLE CLASSES

box.classList.add("one", "two", "three")

box.classList.remove("one", "two")

TOGGLE WITH FORCE

Force add or remove:

box.classList.toggle("active", true)

box.classList.toggle("active", false)

let isLarge = someCondition

box.classList.toggle("large", isLarge)

REPLACE A CLASS

box.classList.replace("old-class", "new-class")

Returns true if the old class existed, false otherwise.

ITERATING CLASSES

let box = document.getElementById("box")

box.className = "one two three"

console.log(box.classList.length)

```
for (let className of box.classList) {
 console.log(className)
}
```

Output:
3
one
two

three

CSS VARIABLES

Set CSS custom properties:

CSS:

```
:root {
 --primary-color: blue;
 --spacing: 20px;
}
```

JavaScript:

```
document.documentElement.style.setProperty("--primary-color",
"red")

document.documentElement.style.setProperty("--spacing", "30px")
```

Read CSS variables:

```
let root = getComputedStyle(document.documentElement)
let primaryColor = root.getPropertyValue("--primary-color")

console.log(primaryColor)
```

STYLING PATTERNS

Theme switching:

```
function setTheme(theme) {
 document.body.className = theme
}
```

```
setTheme("dark-theme")
setTheme("light-theme")
```

Or with data attribute:

```
function setTheme(theme) {
 document.body.dataset.theme = theme
}
```

CSS:

```
[data-theme="dark"] {
 background: #1a1a1a;
 color: white;
}

[data-theme="light"] {
 background: white;
 color: black;
}
```

Loading state:

```
function setLoading(isLoading) {
 let button = document.getElementById("submit")

 button.classList.toggle("loading", isLoading)
 button.disabled = isLoading
 button.textContent = isLoading ? "Loading..." : "Submit"
}
```

Error state:

```
function showError(inputId, message) {
 let input = document.getElementById(inputId)
 let errorSpan = input.nextElementSibling

 input.classList.add("error")
 errorSpan.textContent = message
```

```
 errorSpan.style.display = "block"
}

function clearError(inputId) {
 let input = document.getElementById(inputId)
 let errorSpan = input.nextElementSibling

 input.classList.remove("error")
 errorSpan.textContent = ""
 errorSpan.style.display = "none"
}
```

ANIMATION WITH CLASSES

CSS:

```
.box {
 transition: all 0.3s ease;
}

.box.expanded {
 transform: scale(1.2);
 background: blue;
}
```

JavaScript:

```
let box = document.getElementById("box")

box.classList.toggle("expanded")
```

Let CSS handle animations. JavaScript just toggles classes.

PRACTICAL EXAMPLES

Accordion:

```
function toggleAccordion(header) {
 let content = header.nextElementSibling
 let isOpen = header.classList.contains("open")

 document.querySelectorAll(".accordion-header").forEach(h => {
 h.classList.remove("open")
 h.nextElementSibling.style.maxHeight = "0"
 })

 if (!isOpen) {
 header.classList.add("open")
 content.style.maxHeight = content.scrollHeight + "px"
 }
}
```

Modal:

```
function openModal(modalId) {
 let modal = document.getElementById(modalId)
 modal.classList.add("active")
 document.body.classList.add("modal-open")
}

function closeModal(modalId) {
 let modal = document.getElementById(modalId)
 modal.classList.remove("active")
 document.body.classList.remove("modal-open")
}
```

Highlight on scroll:

```
function highlightOnScroll() {
 let sections = document.querySelectorAll("section")

 sections.forEach(section => {
 let rect = section.getBoundingClientRect()
 let isVisible = rect.top < window.innerHeight && rect.bottom >
```

```
 section.classList.toggle("visible", isVisible)
 })
}

window.addEventListener("scroll", highlightOnScroll)
```

PRACTICE

Set multiple styles using the style property.

Toggle a class on click.

Read computed styles from an element.

Create a theme switcher using classes.

Use CSS variables with JavaScript.

WHAT YOU LEARNED

You learned style property for inline styles.
You learned CSS names convert to camelCase.
You learned getComputedStyle reads actual styles.
You learned classList for adding and removing classes.
You learned toggle, contains, and replace methods.
You learned CSS variables can be set from JavaScript.

The next chapter teaches you about DOM traversal.

Chapter 6: DOM Traversal

NAVIGATING THE DOM TREE

Elements are connected in a tree. You can navigate from one element to its relatives.

HTML:

```
<div id="container">
 <h1>Title</h1>
 <p>First paragraph</p>
 <p>Second paragraph</p>
</div>
```

The container is the parent.
The h1 and p elements are children.
The h1 and p elements are siblings.

PARENT ELEMENT

Get the parent:

```
let paragraph = document.querySelector("p")

let parent = paragraph.parentElement

console.log(parent.id)
```

Output: container

parentElement returns the parent element or null if none.

PARENTNODE VS PARENTELEMENT

```
let element = document.querySelector("p")

console.log(element.parentNode)
console.log(element.parentElement)
```

Usually they return the same thing.

Difference: parentNode can return non-element nodes (like document).
parentElement only returns elements.

For most cases, use parentElement.

CHILDREN

Get all child elements:

```
let container = document.getElementById("container")

let children = container.children

console.log(children.length)

for (let child of children) {
 console.log(child.tagName)
}
```

Output:
3
H1
P
P

children returns only element nodes.

CHILDNODES VS CHILDREN

```
let container = document.getElementById("container")

console.log(container.childNodes.length)
console.log(container.children.length)
```

childNodes includes text nodes and comment nodes.
children includes only element nodes.

HTML:

```
<div id="container">
 <h1>Title</h1>
 <p>Paragraph</p>
</div>
```

childNodes includes the whitespace text nodes between elements.
children includes only h1 and p.

Use children for working with elements.

FIRST AND LAST CHILD

```
let container = document.getElementById("container")

let first = container.firstElementChild
let last = container.lastElementChild

console.log(first.tagName)
console.log(last.tagName)
```

Output:
H1

P

Use firstElementChild and lastElementChild for elements.
Use firstChild and lastChild for any node (including text).

SIBLINGS

Get adjacent siblings:

```
let middle = document.querySelectorAll("p")[0]

let previous = middle.previousElementSibling
let next = middle.nextElementSibling

console.log(previous.tagName)
console.log(next.tagName)
```

Output:
H1
P

Returns null if no sibling exists:

```
let first = document.querySelector("h1")

console.log(first.previousElementSibling)
```

Output: null

TRAVERSING UP THE TREE

Find ancestor with closest:

HTML:

```
<div class="card">
 <div class="card-body">
 <button class="delete">Delete</button>
 </div>
</div>
```

JavaScript:

```
let button = document.querySelector(".delete")

let card = button.closest(".card")

console.log(card.className)
```

Output: card

closest searches up through ancestors for a matching selector.

Returns null if no match:

```
let form = button.closest("form")
console.log(form)
```

Output: null

TRAVERSING DOWN THE TREE

querySelector and querySelectorAll work on any element:

```
let container = document.getElementById("container")

let firstP = container.querySelector("p")
let allPs = container.querySelectorAll("p")

console.log(firstP.textContent)
console.log(allPs.length)
```

This searches only within the container.

CHECKING RELATIONSHIPS

Check if one element contains another:

```
let container = document.getElementById("container")
let paragraph = document.querySelector("p")
let outsideElement = document.getElementById("footer")

console.log(container.contains(paragraph))
console.log(container.contains(outsideElement))
```

Output:
true
false

MATCHES

Check if element matches a selector:

```
let paragraph = document.querySelector("p")

console.log(paragraph.matches("p"))
console.log(paragraph.matches(".intro"))
console.log(paragraph.matches("div p"))
```

Output:
true
false
true (if paragraph is inside a div)

WALKING THE ENTIRE TREE

Visit every element:

```
function walkTree(element, callback) {
 callback(element)

 for (let child of element.children) {
 walkTree(child, callback)
 }
}

walkTree(document.body, element = > {
 console.log(element.tagName)
})
```

This prints every element tag in the document.

PRACTICAL EXAMPLES

Find all parent elements:

```
function getAncestors(element) {
 let ancestors = []
 let current = element.parentElement

 while (current) {
 ancestors.push(current)
 current = current.parentElement
 }

 return ancestors
}

let button = document.querySelector("button")
```

```
let ancestors = getAncestors(button)

ancestors.forEach(el => console.log(el.tagName))
```

Find all siblings:

```
function getSiblings(element) {
 let siblings = []
 let parent = element.parentElement

 if (!parent) return siblings

 for (let child of parent.children) {
 if (child !== element) {
 siblings.push(child)
 }
 }

 return siblings
}
```

Delete button that removes its card:

```
let deleteButtons = document.querySelectorAll(".delete-card")

deleteButtons.forEach(button => {
 button.onclick = function() {
 let card = this.closest(".card")
 card.remove()
 }
})
```

Highlight parent on hover:

```
let items = document.querySelectorAll(".list-item")

items.forEach(item => {
 item.onmouseenter = function() {
 this.parentElement.classList.add("highlight")
 }
```

```
 item.onmouseleave = function() {
 this.parentElement.classList.remove("highlight")
 }
})
```

Navigate between sections:

```
let sections = document.querySelectorAll("section")

function goToNextSection(current) {
 let next = current.nextElementSibling

 if (next && next.tagName === "SECTION") {
 next.scrollIntoView({ behavior: "smooth" })
 }
}

function goToPreviousSection(current) {
 let prev = current.previousElementSibling

 if (prev && prev.tagName === "SECTION") {
 prev.scrollIntoView({ behavior: "smooth" })
 }
}
```

DOM TRAVERSAL SUMMARY

Up:
parentElement - direct parent
closest(selector) - nearest matching ancestor

Down:
children - direct child elements
firstElementChild - first child element
lastElementChild - last child element
querySelector(selector) - first matching descendant

querySelectorAll(selector) - all matching descendants

Sideways:
previousElementSibling - previous sibling element
nextElementSibling - next sibling element

Checks:
contains(element) - is element a descendant?
matches(selector) - does element match selector?

PRACTICE

Navigate from a child to its parent.

Get all siblings of an element.

Use closest to find an ancestor.

Walk through all descendants of an element.

Build a function using multiple traversal methods.

WHAT YOU LEARNED

You learned parentElement gets the parent.
You learned children gets child elements.
You learned firstElementChild and lastElementChild.
You learned previousElementSibling and nextElementSibling.
You learned closest finds ancestors matching a selector.
You learned contains and matches for checking relationships.

The next chapter teaches you about forms and input handling.
BOOK 7: THE DOM
Chapter 7: Forms and Input

ACCESSING FORMS

HTML:

```
<form id="login-form">
 <input type="text" name="username">
 <input type="password" name="password">
 <button type="submit">Login</button>
</form>
```

JavaScript:

```
let form = document.getElementById("login-form")

let form = document.forms[0]

let form = document.forms["login-form"]

console.log(form)
```

ACCESSING FORM ELEMENTS

Access inputs by name:

```
let form = document.getElementById("login-form")

let username = form.elements.username
let password = form.elements.password

console.log(username.value)
console.log(password.value)
```

Or use form.elements with index:

```
let firstInput = form.elements[0]
let secondInput = form.elements[1]
```

INPUT VALUES

Text input:

```
let nameInput = document.getElementById("name")

console.log(nameInput.value)

nameInput.value = "Alice"
```

Number input:

```
let ageInput = document.getElementById("age")

let age = parseInt(ageInput.value)

let price = parseFloat(priceInput.value)
```

CHECKBOX

HTML:

```
<input type="checkbox" id="agree" name="agree">
```

JavaScript:

```
let checkbox = document.getElementById("agree")

console.log(checkbox.checked)
```

```
checkbox.checked = true
checkbox.checked = false

if (checkbox.checked) {
 console.log("User agreed")
}
```

RADIO BUTTONS

HTML:

```
<input type="radio" name="color" value="red"> Red
<input type="radio" name="color" value="green"> Green
<input type="radio" name="color" value="blue"> Blue
```

JavaScript:

```
let radios = document.querySelectorAll("input[name='color']")

let selected = null

radios.forEach(radio => {
 if (radio.checked) {
 selected = radio.value
 }
})

console.log(selected)
```

Or use form.elements:

```
let form = document.getElementById("my-form")
let colorValue = form.elements.color.value

console.log(colorValue)
```

SELECT DROPDOWN

HTML:

```
<select id="country">
 <option value="">Select a country</option>
 <option value="us">United States</option>
 <option value="uk">United Kingdom</option>
 <option value="ca">Canada</option>
</select>
```

JavaScript:

```
let select = document.getElementById("country")

console.log(select.value)

select.value = "uk"
```

Get selected option:

```
let selectedOption = select.options[select.selectedIndex]

console.log(selectedOption.text)
console.log(selectedOption.value)
```

MULTIPLE SELECT

HTML:

```
<select id="languages" multiple>
 <option value="js">JavaScript</option>
 <option value="py">Python</option>
 <option value="java">Java</option>
```

```
</select>
```

JavaScript:

```
let select = document.getElementById("languages")

let selected = []

for (let option of select.options) {
 if (option.selected) {
 selected.push(option.value)
 }
}

console.log(selected)
```

Or use selectedOptions:

```
let selected = Array.from(select.selectedOptions).map(opt =>
opt.value)
```

TEXTAREA

HTML:

```
<textarea id="message" rows="4"></textarea>
```

JavaScript:

```
let textarea = document.getElementById("message")

console.log(textarea.value)

textarea.value = "Hello, world!"

console.log(textarea.value.length)
```

FILE INPUT

HTML:

```
<input type="file" id="avatar">
```

JavaScript:

```
let fileInput = document.getElementById("avatar")

let files = fileInput.files

if (files.length > 0) {
 let file = files[0]

 console.log(file.name)
 console.log(file.size)
 console.log(file.type)
}
```

Multiple files:

HTML:

```
<input type="file" id="photos" multiple>
```

JavaScript:

```
let fileInput = document.getElementById("photos")

for (let file of fileInput.files) {
 console.log(file.name)
}
```

FORM VALIDATION

HTML5 validation attributes:

```
<input type="email" required>
<input type="text" minlength="3" maxlength="20">
<input type="number" min="0" max="100">
<input type="text" pattern="[A-Za-z]+">
```

Check validity in JavaScript:

```
let input = document.getElementById("email")

console.log(input.validity.valid)
console.log(input.validity.valueMissing)
console.log(input.validity.typeMismatch)
```

Check if entire form is valid:

```
let form = document.getElementById("my-form")

if (form.checkValidity()) {
 console.log("Form is valid")
} else {
 console.log("Form has errors")
}
```

CUSTOM VALIDATION

Set custom error message:

```
let input = document.getElementById("username")

if (input.value.length < 3) {
 input.setCustomValidity("Username must be at least 3 characters")
} else {
 input.setCustomValidity("")
```

}

Show validation message:

input.reportValidity()

FOCUS AND BLUR

Focus an input:

let input = document.getElementById("name")

input.focus()

Remove focus:

input.blur()

Check if focused:

```
if (document.activeElement === input) {
 console.log("Input is focused")
}
```

SELECT TEXT

Select all text in an input:

let input = document.getElementById("name")

input.select()

Select a range:

```
input.setSelectionRange(0, 5)
```

DISABLING INPUTS

```
let button = document.getElementById("submit")

button.disabled = true

button.disabled = false

let input = document.getElementById("name")
input.disabled = true
```

READING ALL FORM DATA

FormData object:

```
let form = document.getElementById("my-form")
let formData = new FormData(form)

for (let [name, value] of formData) {
 console.log(`${name}: ${value}`)
}
```

Convert to object:

```
let form = document.getElementById("my-form")
let formData = new FormData(form)
let data = Object.fromEntries(formData)

console.log(data)
```

PRACTICAL EXAMPLES

Login form validation:

```
function validateLoginForm() {
 let form = document.getElementById("login-form")
 let username = form.elements.username.value.trim()
 let password = form.elements.password.value

 let errors = []

 if (username.length < 3) {
 errors.push("Username must be at least 3 characters")
 }

 if (password.length < 8) {
 errors.push("Password must be at least 8 characters")
 }

 return errors
}
```

Show form errors:

```
function showErrors(errors) {
 let errorContainer = document.getElementById("errors")
 errorContainer.innerHTML = ""

 if (errors.length === 0) {
 errorContainer.style.display = "none"
 return
 }

 errorContainer.style.display = "block"

 errors.forEach(error => {
 let p = document.createElement("p")
 p.textContent = error
 p.className = "error"
```

```
 errorContainer.appendChild(p)
 })
}
```

Clear form:

```
function clearForm(formId) {
 let form = document.getElementById(formId)
 form.reset()
}
```

Auto-save form:

```
function autoSave(formId) {
 let form = document.getElementById(formId)
 let formData = new FormData(form)
 let data = Object.fromEntries(formData)

 localStorage.setItem(formId, JSON.stringify(data))
}

function loadSaved(formId) {
 let saved = localStorage.getItem(formId)

 if (saved) {
 let data = JSON.parse(saved)
 let form = document.getElementById(formId)

 for (let [name, value] of Object.entries(data)) {
 if (form.elements[name]) {
 form.elements[name].value = value
 }
 }
 }
}
```

PRACTICE

Get the value of a text input.

Check if a checkbox is checked.

Get the selected value from a dropdown.

Validate a form and show errors.

Use FormData to read all form values.

WHAT YOU LEARNED

You learned to access forms and form elements.
You learned value property for inputs.
You learned checked property for checkboxes.
You learned to work with select dropdowns.
You learned HTML5 validation and custom validation.
You learned FormData for reading all form values.

The next chapter teaches you how to build interactive components.

Chapter 8: Building Interactive Components

BUILDING A TOGGLE SWITCH

HTML:

```
<div class="toggle-container">
  <button id="toggle" class="toggle">OFF</button>
</div>
```

CSS:

```
.toggle {
 padding: 10px 20px;
 font-size: 16px;
 border: none;
 border-radius: 5px;
 cursor: pointer;
 background: #ccc;
 color: black;
 transition: all 0.3s;
}

.toggle.active {
 background: #4CAF50;
 color: white;
}
```

JavaScript:

```
let toggle = document.getElementById("toggle")

toggle.onclick = function() {
 this.classList.toggle("active")
 this.textContent = this.classList.contains("active") ? "ON" : "OFF"
}
```

BUILDING TABS

HTML:

```
<div class="tabs">
 <button class="tab active" data-tab="home">Home</button>
 <button class="tab" data-tab="about">About</button>
 <button class="tab" data-tab="contact">Contact</button>
</div>

<div class="tab-content active" data-content="home">
```

```
 <h2>Home</h2>
 <p>Welcome to the home page.</p>
</div>

<div class="tab-content" data-content="about">
 <h2>About</h2>
 <p>Learn more about us.</p>
</div>

<div class="tab-content" data-content="contact">
 <h2>Contact</h2>
 <p>Get in touch with us.</p>
</div>
```

CSS:

```
.tab-content {
 display: none;
}

.tab-content.active {
 display: block;
}

.tab.active {
 background: #4CAF50;
 color: white;
}
```

JavaScript:

```
let tabs = document.querySelectorAll(".tab")

tabs.forEach(tab => {
 tab.onclick = function() {
 let targetTab = this.dataset.tab

 document.querySelectorAll(".tab").forEach(t =>
t.classList.remove("active"))
 document.querySelectorAll(".tab-content").forEach(c =>
```

```
c.classList.remove("active"))

 this.classList.add("active")
 document.querySelector(`[data-
content="${targetTab}"]`).classList.add("active")
 }
})
```

BUILDING AN ACCORDION

HTML:

```
<div class="accordion">
 <div class="accordion-item">
 <button class="accordion-header">Section 1</button>
 <div class="accordion-content">
 <p>Content for section 1.</p>
 </div>
 </div>

 <div class="accordion-item">
 <button class="accordion-header">Section 2</button>
 <div class="accordion-content">
 <p>Content for section 2.</p>
 </div>
 </div>

 <div class="accordion-item">
 <button class="accordion-header">Section 3</button>
 <div class="accordion-content">
 <p>Content for section 3.</p>
 </div>
 </div>
</div>
```

CSS:

```
.accordion-content {
 max-height: 0;
 overflow: hidden;
 transition: max-height 0.3s ease;
}

.accordion-header.active + .accordion-content {
 max-height: 200px;
}
```

JavaScript:

```
let headers = document.querySelectorAll(".accordion-header")

headers.forEach(header => {
 header.onclick = function() {
 let isOpen = this.classList.contains("active")

 document.querySelectorAll(".accordion-header").forEach(h => {
 h.classList.remove("active")
 })

 if (!isOpen) {
 this.classList.add("active")
 }
 }
})
```

BUILDING A MODAL

HTML:

```
<button id="open-modal">Open Modal</button>

<div id="modal" class="modal">
 <div class="modal-content">
 <button class="modal-close">&times;</button>
```

```
  <h2>Modal Title</h2>
  <p>This is the modal content.</p>
  </div>
</div>
```

CSS:

```
.modal {
 display: none;
 position: fixed;
 top: 0;
 left: 0;
 width: 100%;
 height: 100%;
 background: rgba(0, 0, 0, 0.5);
 justify-content: center;
 align-items: center;
}

.modal.active {
 display: flex;
}

.modal-content {
 background: white;
 padding: 30px;
 border-radius: 10px;
 max-width: 500px;
 position: relative;
}

.modal-close {
 position: absolute;
 top: 10px;
 right: 10px;
 font-size: 24px;
 border: none;
 background: none;
 cursor: pointer;
}
```

JavaScript:

```
let openBtn = document.getElementById("open-modal")
let modal = document.getElementById("modal")
let closeBtn = modal.querySelector(".modal-close")

openBtn.onclick = function() {
 modal.classList.add("active")
}

closeBtn.onclick = function() {
 modal.classList.remove("active")
}

modal.onclick = function(e) {
 if (e.target === modal) {
 modal.classList.remove("active")
 }
}
```

BUILDING A DROPDOWN MENU

HTML:

```
<div class="dropdown">
 <button class="dropdown-toggle">Select Option</button>
 <ul class="dropdown-menu">
 <li data-value="option1">Option 1</li>
 <li data-value="option2">Option 2</li>
 <li data-value="option3">Option 3</li>
 </ul>
</div>
```

CSS:

```
.dropdown {
```

```
 position: relative;
 display: inline-block;
}

.dropdown-menu {
 display: none;
 position: absolute;
 top: 100%;
 left: 0;
 background: white;
 border: 1px solid #ccc;
 list-style: none;
 padding: 0;
 margin: 0;
 min-width: 150px;
}

.dropdown.open .dropdown-menu {
 display: block;
}

.dropdown-menu li {
 padding: 10px;
 cursor: pointer;
}

.dropdown-menu li:hover {
 background: #f0f0f0;
}
```

JavaScript:

```
let dropdown = document.querySelector(".dropdown")
let toggle = dropdown.querySelector(".dropdown-toggle")
let items = dropdown.querySelectorAll(".dropdown-menu li")

toggle.onclick = function() {
 dropdown.classList.toggle("open")
}
```

```
items.forEach(item => {
 item.onclick = function() {
 toggle.textContent = this.textContent
 dropdown.dataset.value = this.dataset.value
 dropdown.classList.remove("open")
 }
})

document.onclick = function(e) {
 if (!dropdown.contains(e.target)) {
 dropdown.classList.remove("open")
 }
}
```

BUILDING A TODO LIST

HTML:

```
<div class="todo-app">
 <input type="text" id="todo-input" placeholder="Add a task">
 <button id="add-todo">Add</button>
 <ul id="todo-list"></ul>
</div>
```

JavaScript:

```
let input = document.getElementById("todo-input")
let addBtn = document.getElementById("add-todo")
let list = document.getElementById("todo-list")

function addTodo() {
 let text = input.value.trim()

 if (text === "") return

 let li = document.createElement("li")
```

```
 let checkbox = document.createElement("input")
 checkbox.type = "checkbox"
 checkbox.onchange = function() {
 li.classList.toggle("completed", this.checked)
 }

 let span = document.createElement("span")
 span.textContent = text

 let deleteBtn = document.createElement("button")
 deleteBtn.textContent = "Delete"
 deleteBtn.onclick = function() {
 li.remove()
 }

 li.appendChild(checkbox)
 li.appendChild(span)
 li.appendChild(deleteBtn)
 list.appendChild(li)

 input.value = ""
 input.focus()
}

addBtn.onclick = addTodo

input.onkeypress = function(e) {
 if (e.key === "Enter") {
 addTodo()
 }
}
```

BUILDING AN IMAGE CAROUSEL

HTML:

```
<div class="carousel">
```

```
 <button class="prev">&lt;</button>
 <div class="slides">
 <img src="image1.jpg" class="slide active">
 <img src="image2.jpg" class="slide">
 <img src="image3.jpg" class="slide">
 </div>
 <button class="next">&gt;</button>
</div>
```

CSS:

```
.carousel {
 display: flex;
 align-items: center;
}

.slides {
 position: relative;
 width: 400px;
 height: 300px;
}

.slide {
 position: absolute;
 width: 100%;
 height: 100%;
 object-fit: cover;
 opacity: 0;
 transition: opacity 0.5s;
}

.slide.active {
 opacity: 1;
}
```

JavaScript:

```
let slides = document.querySelectorAll(".slide")
let currentIndex = 0
```

```
function showSlide(index) {
 slides.forEach(slide => slide.classList.remove("active"))

 if (index >= slides.length) index = 0
 if (index < 0) index = slides.length - 1

 currentIndex = index
 slides[currentIndex].classList.add("active")
}

document.querySelector(".prev").onclick = function() {
 showSlide(currentIndex - 1)
}

document.querySelector(".next").onclick = function() {
 showSlide(currentIndex + 1)
}

setInterval(() => {
 showSlide(currentIndex + 1)
}, 5000)
```

BUILDING A COUNTER

HTML:

```
<div class="counter">
 <button id="decrement">-</button>
 <span id="count">0</span>
 <button id="increment">+</button>
</div>
```

JavaScript:

```
let count = 0
let countDisplay = document.getElementById("count")
```

```
document.getElementById("decrement").onclick = function() {
 count--
 countDisplay.textContent = count
}

document.getElementById("increment").onclick = function() {
 count++
 countDisplay.textContent = count
}
```

PRACTICE

Build a toggle button that switches between states.

Create a tab component that shows different content.

Build a modal with open and close functionality.

Create a dropdown menu that closes when clicking outside.

Build a todo list with add and delete functionality.

WHAT YOU LEARNED

You learned to build toggle switches.
You learned to create tab interfaces.
You learned to build accordion components.
You learned to create modal dialogs.
You learned to build dropdown menus.
You learned to create interactive todo lists.

The next chapter teaches you about animations and transitions.

BOOK 7: THE DOM

Chapter 9: Animations and Transitions

CSS TRANSITIONS

CSS transitions animate property changes:

CSS:

```
.box {
 width: 100px;
 height: 100px;
 background: blue;
 transition: all 0.3s ease;
}

.box.expanded {
 width: 200px;
 height: 200px;
 background: red;
}
```

JavaScript:

```
let box = document.querySelector(".box")

box.onclick = function() {
 this.classList.toggle("expanded")
}
```

The transition happens automatically when the class changes.

TRANSITION PROPERTIES

CSS:

```
.element {
 transition-property: background-color, transform;
 transition-duration: 0.3s;
 transition-timing-function: ease-in-out;
 transition-delay: 0.1s;
}

.element {
 transition: background-color 0.3s ease-in-out 0.1s, transform 0.3s
ease;
}

.element {
 transition: all 0.3s ease;
}
```

Timing functions:
ease - slow start, fast middle, slow end
linear - constant speed
ease-in - slow start
ease-out - slow end
ease-in-out - slow start and end

TRIGGERING TRANSITIONS WITH JAVASCRIPT

Add or remove classes:

```
element.classList.add("fade-in")
element.classList.remove("fade-out")
element.classList.toggle("visible")
```

Change styles directly:

```
element.style.opacity = "1"
element.style.transform = "translateX(100px)"
```

The CSS transition handles the animation.

DETECTING TRANSITION END

Wait for transition to complete:

```
let box = document.querySelector(".box")

box.classList.add("moving")

box.addEventListener("transitionend", function(e) {
 console.log("Transition complete")
 console.log(e.propertyName)
})
```

transitionend fires for each property that transitions.

CSS ANIMATIONS

CSS:

```
@keyframes fadeIn {
 from {
 opacity: 0;
 }
 to {
 opacity: 1;
 }
}

@keyframes slideIn {
 0% {
 transform: translateX(-100%);
 opacity: 0;
```

```
 }
 100% {
 transform: translateX(0);
 opacity: 1;
 }
}

.fade-in {
 animation: fadeIn 0.5s ease forwards;
}

.slide-in {
 animation: slideIn 0.5s ease forwards;
}
```

JavaScript:

```
let element = document.querySelector(".element")

element.classList.add("fade-in")
```

DETECTING ANIMATION END

```
let element = document.querySelector(".element")

element.classList.add("bounce")

element.addEventListener("animationend", function(e) {
 console.log("Animation complete")
 console.log(e.animationName)

 this.classList.remove("bounce")
})
```

REQUESTANIMATIONFRAME

For smooth custom animations:

```
let box = document.querySelector(".box")
let position = 0

function animate() {
 position + = 2

 box.style.transform = `translateX(${position}px)`

 if (position < 300) {
 requestAnimationFrame(animate)
 }
}

requestAnimationFrame(animate)
```

requestAnimationFrame runs before the next repaint (usually 60fps).

ANIMATION WITH TIMING

```
function animate(duration, callback) {
 let start = performance.now()

 function step(timestamp) {
 let progress = (timestamp - start) / duration

 if (progress > 1) progress = 1

 callback(progress)

 if (progress < 1) {
 requestAnimationFrame(step)
 }
```

```
 }

 requestAnimationFrame(step)
}

animate(1000, progress => {
 let box = document.querySelector(".box")
 box.style.transform = `translateX(${progress * 300}px)`
})
```

EASING FUNCTIONS

Linear animation feels mechanical. Easing makes it natural.

```
function easeInOut(t) {
 return t < 0.5 ? 2 * t * t : -1 + (4 - 2 * t) * t
}

function easeOut(t) {
 return 1 - Math.pow(1 - t, 3)
}

function easeIn(t) {
 return t * t * t
}

animate(1000, progress => {
 let eased = easeOut(progress)
 box.style.transform = `translateX(${eased * 300}px)`
})
```

FADE IN AND OUT

```
function fadeIn(element, duration = 300) {
```

```
 element.style.opacity = "0"
 element.style.display = "block"

 let start = performance.now()

 function step(timestamp) {
 let progress = (timestamp - start) / duration

 if (progress > 1) progress = 1

 element.style.opacity = progress

 if (progress < 1) {
 requestAnimationFrame(step)
 }
 }

 requestAnimationFrame(step)
}

function fadeOut(element, duration = 300) {
 let start = performance.now()

 function step(timestamp) {
 let progress = (timestamp - start) / duration

 if (progress > 1) progress = 1

 element.style.opacity = 1 - progress

 if (progress < 1) {
 requestAnimationFrame(step)
 } else {
 element.style.display = "none"
 }
 }

 requestAnimationFrame(step)
}
```

SLIDE TOGGLE

```
function slideDown(element, duration = 300) {
 element.style.display = "block"
 let height = element.scrollHeight

 element.style.height = "0"
 element.style.overflow = "hidden"

 let start = performance.now()

 function step(timestamp) {
 let progress = (timestamp - start) / duration

 if (progress > 1) progress = 1

 element.style.height = (progress * height) + "px"

 if (progress < 1) {
 requestAnimationFrame(step)
 } else {
 element.style.height = ""
 element.style.overflow = ""
 }
 }

 requestAnimationFrame(step)
}

function slideUp(element, duration = 300) {
 let height = element.scrollHeight

 element.style.height = height + "px"
 element.style.overflow = "hidden"

 let start = performance.now()
```

```
 function step(timestamp) {
 let progress = (timestamp - start) / duration

 if (progress > 1) progress = 1

 element.style.height = ((1 - progress) * height) + "px"

 if (progress < 1) {
 requestAnimationFrame(step)
 } else {
 element.style.display = "none"
 element.style.height = ""
 element.style.overflow = ""
 }
 }

 requestAnimationFrame(step)
}
```

SCROLL ANIMATIONS

Animate elements as they scroll into view:

```
function isInViewport(element) {
 let rect = element.getBoundingClientRect()

 return (
 rect.top < window.innerHeight &&
 rect.bottom > 0
 )
}

function handleScroll() {
 let elements = document.querySelectorAll(".animate-on-scroll")

 elements.forEach(element => {
 if (isInViewport(element)) {
```

```
 element.classList.add("visible")
 }
 })
}

window.addEventListener("scroll", handleScroll)

handleScroll()
```

CSS:

```
.animate-on-scroll {
 opacity: 0;
 transform: translateY(50px);
 transition: all 0.6s ease;
}

.animate-on-scroll.visible {
 opacity: 1;
 transform: translateY(0);
}
```

SMOOTH SCROLLING

Scroll to element:

```
function scrollToElement(element) {
 element.scrollIntoView({
 behavior: "smooth",
 block: "start"
 })
}

let section = document.getElementById("about")
scrollToElement(section)
```

Scroll to top:

```
function scrollToTop() {
 window.scrollTo({
 top: 0,
 behavior: "smooth"
 })
}
```

PRACTICAL EXAMPLES

Notification with auto-dismiss:

```
function showNotification(message, duration = 3000) {
 let notification = document.createElement("div")
 notification.className = "notification"
 notification.textContent = message

 document.body.appendChild(notification)

 requestAnimationFrame(() => {
 notification.classList.add("show")
 })

 setTimeout(() => {
 notification.classList.remove("show")

 notification.addEventListener("transitionend", () => {
 notification.remove()
 })
 }, duration)
}
```

CSS:

```
.notification {
 position: fixed;
 top: 20px;
```

```
 right: 20px;
 background: #333;
 color: white;
 padding: 15px 20px;
 border-radius: 5px;
 transform: translateX(100%);
 transition: transform 0.3s ease;
}

.notification.show {
 transform: translateX(0);
}
```

Loading spinner:

HTML:

```
<div class="spinner" id="loader"></div>
```

CSS:

```
@keyframes spin {
 0% { transform: rotate(0deg); }
 100% { transform: rotate(360deg); }
}

.spinner {
 width: 40px;
 height: 40px;
 border: 4px solid #f3f3f3;
 border-top: 4px solid #3498db;
 border-radius: 50%;
 animation: spin 1s linear infinite;
}
```

JavaScript:

```
function showLoader() {
 document.getElementById("loader").style.display = "block"
}
```

```
function hideLoader() {
 document.getElementById("loader").style.display = "none"
}
```

PRACTICE

Create a fade-in animation using CSS transitions.

Build a slide toggle using JavaScript.

Use requestAnimationFrame for a custom animation.

Create a scroll-triggered animation.

Build a notification that animates in and out.

WHAT YOU LEARNED

You learned CSS transitions animate property changes.
You learned CSS animations use keyframes.
You learned requestAnimationFrame for custom animations.
You learned to detect animation and transition end.
You learned to create easing functions.
You learned scroll-triggered animations.

The next chapter combines everything into practice programs.

INTERACTIVE QUIZ APPLICATION

Complete quiz with scoring:

HTML:

```
<div id="quiz-container">
 <div id="quiz">
 <h2 id="question"></h2>
 <div id="answers"></div>
 </div>
 <div id="results" style="display: none;">
 <h2>Quiz Complete!</h2>
 <p id="score"></p>
 <button id="restart">Try Again</button>
 </div>
</div>
```

JavaScript:

```
let quizData = [
 {
 question: "What does HTML stand for?",
 answers: [
 "Hyper Text Markup Language",
 "Home Tool Markup Language",
 "Hyperlinks Text Mark Language"
 ],
 correct: 0
 },
 {
 question: "What does CSS stand for?",
 answers: [
 "Creative Style Sheets",
 "Cascading Style Sheets",
 "Computer Style Sheets"
 ],
 correct: 1
 },
 {
```

```
 question: "What year was JavaScript created?",
 answers: [
 "1995",
 "1990",
 "2000"
 ],
 correct: 0
 }
]

let currentQuestion = 0
let score = 0

let questionEl = document.getElementById("question")
let answersEl = document.getElementById("answers")
let quizEl = document.getElementById("quiz")
let resultsEl = document.getElementById("results")
let scoreEl = document.getElementById("score")
let restartBtn = document.getElementById("restart")

function showQuestion() {
 let data = quizData[currentQuestion]

 questionEl.textContent = data.question
 answersEl.innerHTML = ""

 data.answers.forEach((answer, index) => {
 let button = document.createElement("button")
 button.textContent = answer
 button.className = "answer-btn"
 button.onclick = () => selectAnswer(index)
 answersEl.appendChild(button)
 })
}

function selectAnswer(index) {
 let data = quizData[currentQuestion]

 if (index === data.correct) {
 score++
```

```
 }

 currentQuestion++

 if (currentQuestion < quizData.length) {
 showQuestion()
 } else {
 showResults()
 }
}

function showResults() {
 quizEl.style.display = "none"
 resultsEl.style.display = "block"

 let percentage = Math.round((score / quizData.length) * 100)
 scoreEl.textContent = `You scored ${score} out of
${quizData.length} (${percentage}%)`
}

function restartQuiz() {
 currentQuestion = 0
 score = 0
 quizEl.style.display = "block"
 resultsEl.style.display = "none"
 showQuestion()
}

restartBtn.onclick = restartQuiz

showQuestion()
```

REAL-TIME SEARCH FILTER

Filter a list as you type:

HTML:

```
<input type="text" id="search" placeholder="Search...">
<ul id="list">
 <li>Apple</li>
 <li>Banana</li>
 <li>Cherry</li>
 <li>Date</li>
 <li>Elderberry</li>
 <li>Fig</li>
 <li>Grape</li>
</ul>
```

JavaScript:

```
let searchInput = document.getElementById("search")
let list = document.getElementById("list")
let items = list.querySelectorAll("li")

searchInput.oninput = function() {
 let searchTerm = this.value.toLowerCase()

 items.forEach(item => {
 let text = item.textContent.toLowerCase()

 if (text.includes(searchTerm)) {
 item.style.display = ""
 } else {
 item.style.display = "none"
 }
 })

 let visibleCount = list.querySelectorAll("li:not([style*='display:
none'])").length

 let noResults = document.getElementById("no-results")

 if (visibleCount === 0) {
 if (!noResults) {
 noResults = document.createElement("p")
 noResults.id = "no-results"
```

```
 noResults.textContent = "No results found"
 list.parentNode.insertBefore(noResults, list.nextSibling)
 }
 } else if (noResults) {
 noResults.remove()
 }
}
```

DRAG AND DROP LIST

Reorderable list:

HTML:

```
<ul id="sortable-list">
 <li draggable="true">Item 1</li>
 <li draggable="true">Item 2</li>
 <li draggable="true">Item 3</li>
 <li draggable="true">Item 4</li>
 <li draggable="true">Item 5</li>
</ul>
```

CSS:

```
#sortable-list li {
 padding: 10px;
 margin: 5px 0;
 background: #f0f0f0;
 cursor: move;
}

#sortable-list li.dragging {
 opacity: 0.5;
}

#sortable-list li.dragover {
 border-top: 2px solid blue;
```

```
}
```

JavaScript:

```
let list = document.getElementById("sortable-list")
let items = list.querySelectorAll("li")
let draggingItem = null

items.forEach(item => {
 item.ondragstart = function() {
 draggingItem = this
 this.classList.add("dragging")
 }

 item.ondragend = function() {
 this.classList.remove("dragging")
 draggingItem = null

 list.querySelectorAll("li").forEach(li => {
 li.classList.remove("dragover")
 })
 }

 item.ondragover = function(e) {
 e.preventDefault()

 if (this !== draggingItem) {
 this.classList.add("dragover")
 }
 }

 item.ondragleave = function() {
 this.classList.remove("dragover")
 }

 item.ondrop = function(e) {
 e.preventDefault()
 this.classList.remove("dragover")

 if (this !== draggingItem) {
```

```
let allItems = [...list.querySelectorAll("li")]
let dragIndex = allItems.indexOf(draggingItem)
let dropIndex = allItems.indexOf(this)

if (dragIndex < dropIndex) {
this.parentNode.insertBefore(draggingItem, this.nextSibling)
} else {
this.parentNode.insertBefore(draggingItem, this)
}
}
}
})
```

FORM VALIDATION WITH LIVE FEEDBACK

Validate as user types:

HTML:

```
<form id="signup-form">
<div class="form-group">
<label>Username</label>
<input type="text" id="username" name="username">
<span class="error"></span>
</div>

<div class="form-group">
<label>Email</label>
<input type="email" id="email" name="email">
<span class="error"></span>
</div>

<div class="form-group">
<label>Password</label>
<input type="password" id="password" name="password">
<span class="error"></span>
<div class="password-strength"></div>
```

```
 </div>

 <div class="form-group">
 <label>Confirm Password</label>
 <input type="password" id="confirm" name="confirm">
 <span class="error"></span>
 </div>

 <button type="submit">Sign Up</button>
</form>
```

JavaScript:

```
let form = document.getElementById("signup-form")

let validators = {
 username: {
 validate(value) {
 if (value.length < 3) {
 return "Username must be at least 3 characters"
 }
 if (!/^[a-zA-Z0-9_]+$/.test(value)) {
 return "Username can only contain letters, numbers, and
underscores"
 }
 return ""
 }
 },

 email: {
 validate(value) {
 if (!value.includes("@") || !value.includes(".")) {
 return "Please enter a valid email"
 }
 return ""
 }
 },

 password: {
 validate(value) {
```

```
 if (value.length < 8) {
 return "Password must be at least 8 characters"
 }
 return ""
 }
 },

 confirm: {
 validate(value) {
 let password = document.getElementById("password").value
 if (value !== password) {
 return "Passwords do not match"
 }
 return ""
 }
 }
}

function validateField(input) {
 let name = input.name
 let value = input.value
 let errorSpan = input.nextElementSibling

 let validator = validators[name]
 let error = validator ? validator.validate(value) : ""

 errorSpan.textContent = error

 if (error) {
 input.classList.add("invalid")
 input.classList.remove("valid")
 } else if (value) {
 input.classList.add("valid")
 input.classList.remove("invalid")
 }
}

function updatePasswordStrength(password) {
 let strength = 0
 let strengthBar = document.querySelector(".password-strength")
```

```
 if (password.length >= 8) strength++
 if (/[a-z]/.test(password) && /[A-Z]/.test(password)) strength++
 if (/\d/.test(password)) strength++
 if (/[^a-zA-Z0-9]/.test(password)) strength++

 let labels = ["Weak", "Fair", "Good", "Strong"]
 let colors = ["red", "orange", "yellow", "green"]

 strengthBar.textContent = password ? labels[strength - 1] || "Very
Weak" : ""
 strengthBar.style.color = colors[strength - 1] || "red"
}

let inputs = form.querySelectorAll("input")

inputs.forEach(input => {
 input.oninput = function() {
 validateField(this)

 if (this.name === "password") {
 updatePasswordStrength(this.value)
 let confirm = document.getElementById("confirm")
 if (confirm.value) validateField(confirm)
 }
 }

 input.onblur = function() {
 validateField(this)
 }
})

form.onsubmit = function(e) {
 e.preventDefault()

 let isValid = true

 inputs.forEach(input => {
 validateField(input)
 if (input.classList.contains("invalid")) {
```

```
 isValid = false
 }
 })

 if (isValid) {
 console.log("Form submitted!")
 }
}
```

INFINITE SCROLL

Load more content when scrolling:

HTML:

```
<div id="content">
 <div class="item">Item 1</div>
 <div class="item">Item 2</div>
 <div class="item">Item 3</div>
</div>
<div id="loader" style="display: none;">Loading...</div>
```

JavaScript:

```
let content = document.getElementById("content")
let loader = document.getElementById("loader")
let itemCount = 3
let isLoading = false

function loadMore() {
 if (isLoading) return

 isLoading = true
 loader.style.display = "block"

 setTimeout(() => {
 for (let i = 0; i < 5; i++) {
```

```
 itemCount++
 let item = document.createElement("div")
 item.className = "item"
 item.textContent = `Item ${itemCount}`
 content.appendChild(item)
 }

 loader.style.display = "none"
 isLoading = false
 }, 1000)
}

window.onscroll = function() {
 let scrollTop = window.scrollY
 let windowHeight = window.innerHeight
 let documentHeight = document.documentElement.scrollHeight

 if (scrollTop + windowHeight >= documentHeight - 100) {
 loadMore()
 }
}
```

WHAT YOU BUILT

You built five complete programs:

Interactive quiz with scoring and results
Real-time search filter with no-results handling
Drag and drop reorderable list
Form validation with live feedback and password strength
Infinite scroll content loader

Each program demonstrated:

DOM selection and manipulation
Event handling
Dynamic element creation

User interface patterns
State management

WHAT COMES NEXT

You now understand how JavaScript interacts with web pages:

- Selecting elements (Book 7, Chapter 1)
- Modifying elements (Book 7, Chapter 2)
- Creating and removing elements (Book 7, Chapter 3)
- Attributes and data attributes (Book 7, Chapter 4)
- Styles and classes (Book 7, Chapter 5)
- DOM traversal (Book 7, Chapter 6)
- Forms and input (Book 7, Chapter 7)
- Building components (Book 7, Chapter 8)
- Animations (Book 7, Chapter 9)

Book 8 teaches you about events in depth. You will:

- Handle user interactions
- Understand event flow and bubbling
- Use event delegation
- Work with keyboard and mouse events
- Build more interactive applications

The DOM makes JavaScript visual. Events make it interactive.

You finished Book 7. You can now manipulate web pages. Now we add true interactivity.